

### **Remarks**

The above Amendments and these Remarks are in reply to the Office Action mailed October 14, 2008. Applicants have amended claims 1-2, 6, 20, 33, 35, 38, 52 and 62, and cancelled claims 16, 18, 32 and 57-61.

#### **I. Summary of Interview with Examiner**

Applicants thank the Examiner for conducting an Interview (requested by Applicants) by telephone on Friday, February 6, 2009 to discuss the technology disclosed in this application. The participants in the telephonic interview included the Examiner, Qing Chen, and Scott D. Sanford, attorney of record for Applicants.

In the Interview, claim 1 in the present application were discussed in view of the prior art cited in the pending Office Action. Applicants thank the Examiner and his supervisor for the interview.

#### **II. Response to Claim Objections**

Claims 20-35, 38-57 and 62-65 were objected to because of certain informalities. Applicants have amended independent claims 20, 33, 38, 52 and 62 to correct the informalities objected to by the Examiner. Dependent claims 21-32, 34-35, 39-51, 53-57 and 63-65 were objected to by the Examiner because they depend from an objected independent claim. Thus, Applicants request that the Examiner withdraw the claim objections to claims 20-35, 38-57 and 62-65.

#### **III. Response to Rejection of Claims Under 35 U.S.C. §112**

Claims 20-31 were rejected under 35 U.S.C. §112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which

applicant regards as the invention. Applicants have amended claim 20 to provide a sufficient antecedent basis for the claims term “application objects.” Claims 21-31 depend from claim 20. Because claims 20-31 meet the requirements set forth under 35 U.S.C. 112, second paragraph, Applicants request that the Examiner withdraw this rejection.

#### **IV. Response to Rejection Of Claims Under 35 U.S.C. §102**

Claims 1-2, 4, 6-8, 11-15, 17, 19-23, 25, 27-31, 33, 38-39, 41-42, 44-45, 47-58 and 60-62 were rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Patent No. 7,191,441 (“*Abbott*”).

Claim 1 is not anticipated by *Abbott*. In particular, *Abbott* does not disclose “building an optimized object code file using the serialized representation and the first object code file” in the manner recited in claim 1. The “optimized object code file” recited in claim 1 is built by “identifying each application object in the serialized representation that has a unique identifier referring to an application object in the first object code file,” and “for each identified application object, copying the application object in the initialization code with the same unique identifier to the optimized object code file.” The “optimized object code file” is then loaded into, and executed by, a new runtime environment.

*Abbott* discloses restoring the saved state of an application and executing the saved state by an execution engine of a Java VM. The restore process in *Abbott* is disclosed in col. 17, line 62-col. 18, line 65. In this section of *Abbott*, application objects are not copied from an initialization code to an “optimized object code” based on unique identifiers contained in a serialized representation of the initialization code. *Abbott* discloses reloading the application in the following stages (col. 18, lines 13-65):

The various components of the Java VM are provided with restore commands to restart themselves using the saved information. These components are launched in parallel with the major structures such as the heap and stacks being recreated (as shown in FIG. 8 below), according to the desired timing

between the two (e.g. the CL class loader is launched at the same time as the classes are being reconstructed). Note that in order to optimise the load time, the information stored in the snapshot is preferably structured, taking into account the reload order described below.

FIG. 8 depicts the operation of the restore tool. After it has been invoked (step 810), the information covering the general state of the application, DLLs loaded etc., is retrieved. Each DLL recorded in the save file can then be loaded into memory (step 820). The load address of each DLL is added to the table of loaded DLLs for later use. The heap is loaded next, as a single block (step 830). The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code. Any adjustments needed to heap objects (native libraries) to accommodate DLLS loaded are made now. Note that there is some discretion as to heap size, in that this does not necessarily have to match the original heap size. Rather it may for performance reasons be desirable to set the new heap size to some predefined multiple of the total size of the stored objects to be reconstructed into the heap (clearly the new heap must be large enough to accommodate all the saved objects).

Following the heap, thread information is loaded, and used to reconstruct the C-stack, Java stack and thread state (program counter, registers etc.) (step 840). The DLL information loaded earlier is needed to adjust the addresses in the C-stack. The object table (loaded with the heap information) is utilised to rebuild the object addresses held on the Java stacks. The system threads are now created and the saved thread information used to set their states. As the heap will have been initialised by this stage, it is possible to reconstruct the object addresses on the Java stacks. This leaves the threads in a "ready-to-run" state.

Finally, IO channels and socket connections are restored (step 850). It should be noted that this stage has the possibility of failure due to uncontrollable events (files having been deleted, remote machines no longer being available etc.)

At the conclusion of the above operations, the suspended application is now ready to run, and control is passed to execution engine of the Java VM (step 860). Processing of the application then resumes at the point in the application immediately following the call to the snapshot routine.

As shown above, unlike the method recited in claim 1, *Abbott* reloads a stored application into a Java virtual machine (Java VM) by reloading the various components of

the application (e.g., DLLs, threads, etc.) in stages. *Abbott* does not identify application objects in a serialized representation of a first application state and copy certain application objects into an “optimized object code file.” The table located in col. 17, lines 6-26 of *Abbott* illustrates that the state of an application is saved by serializing the Java VM, compiling any code via a just-in-time (JIT) compiler, determining heap and stack sizes and then taking a snapshot of the application state and Java VM.

The method in claim 1 does not reconstruct application objects or restore the runtime environment to generate the “optimized object code.” As recited in claim 1, application source code is initially compiled into a first object file. Initialization code is created based on the first object code file and is subsequently serialized. Each application object is the serialized representation of the initialization code is marked with a unique identifier. Using the serialized representation, application objects in the initialization code “with the same unique identifier [are copied] to the optimized object code file.” This optimized object code is isomorphic with the initialization code, and can be loaded into, and executed within, a new runtime environment.

*Abbott* does not replicate a first application state by copying application objects from an initialization code. As shown above, *Abbott* must reload each component of the application (application objects) and Java VM (built-in objects) in stages. This is not equivalent to building an “optimized object file” by “identifying each application object in the serialized representation that has a unique identifier referring to an application object in the first object code file,” and “for each identified application object, copying the application object in the initialization code with the same unique identifier to the optimized object code file.” Therefore, the method recited in claim 1 is not anticipated by *Abbott*, and Applicants request that the Examiner withdraw this rejection.

Dependent claims 2, 6-8, 11-15, 17 and 19 depend directly or indirectly from independent claim 1. These dependent claims include all of the limitations of the independent claim from which they depend. Applicants respectfully assert that dependent

claims 2, 6-8, 11-15, 17 and 19 are allowable for at least the reasons set forth above concerning claim 1.

Claim 20 is not anticipated by *Abbott* for at least the same reasons discussed above with regard to claim 1. In particular, *Abbott* does not disclose building an “optimized object code file” by “identifying each application object in the serialized representation that has a unique identifier referring to an application object in the first object code file,” and “for each identified application object, copying the application object in the initialization code with the same unique identifier to the optimized object code file.” Therefore, the method recited in claim 20 is not anticipated by *Abbott*, and Applicants request that the Examiner withdraw this rejection.

Dependent claims 21-23, 25, and 27-31 depend directly or indirectly from independent claim 20. These dependent claims include all of the limitations of the independent claim from which they depend. Applicants respectfully assert that dependent claims 21-23, 25, and 27-31 are allowable for at least the reasons set forth above concerning claim 20.

Claim 33 is not anticipated by *Abbott* for at least the same reasons discussed above regarding claim 1. In particular, *Abbott* does not disclose building an “optimized object code file” by “identifying each application object in the serialized representation that has a unique identifier referring to an application object in the first object code file,” and “for each identified application object, copying the application object in the initialization code with the same unique identifier to the optimized object code file.” Therefore, the method recited in claim 33 is not anticipated by *Abbott*, and Applicants request that the Examiner withdraw this rejection.

Claim 38 is not anticipated by *Abbott* for at least the same reasons discussed above regarding claim 1. In particular, Abbot does not disclose “building an optimized object code file using the serialized representation of the application objects and the first object code file, wherein the optimized object code file includes application objects, each marked with a

unique identifier, so that corresponding application objects in the first application state can be identified, and copying application objects from the second application state to the optimized object code file based on the unique identifiers.” Therefore, claim 38 is not anticipated by *Abbott*, and Applicants request that the Examiner withdraw this rejection.

Dependent claims 39, 41-42, 44-45 and 47-51 depend directly or indirectly from independent claim 38. These dependent claims include all of the limitations of the independent claim from which they depend. Applicants respectfully assert that dependent claims 39, 41-42, 44-45 and 47-51 are allowable for at least the reasons set forth above concerning claim 38.

Claim 52 is not anticipated by *Abbott* for at least the same reasons discussed above regarding claim 1. In particular, *Abbott* does not disclose “building a second object code file using said serialized representation and the first object code file, wherein application objects from the second application state are copied to the second object code file based on a unique identifier associated with each application object in the serialized representation.” Therefore, the method recited in claim 52 is not anticipated by *Abbott*, and Applicants request that the Examiner withdraw this rejection.

Dependent claims 53-56 depend directly or indirectly from independent claim 52. These dependent claims include all of the limitations of the independent claim from which they depend. Applicants respectfully assert that dependent claims 53-56 are allowable for at least the reasons set forth above concerning claim 52.

Claim 62 is not anticipated by *Abbott* for at least the same reasons discussed above regarding claim 1. In particular, *Abbott* does not disclose “building an optimized object code file using the serialized representation and the first object code file, wherein application objects are copied from the third application state into the optimized object code file based on a unique identifier associated with each application object in the serialized representation.” Therefore, the method recited in claim 62 is not anticipated by *Abbott*, and Applicants request that the Examiner withdraw this rejection.

**V.      Response to Rejection Of Claims Under 35 U.S.C. §103**

Claims 3, 40 and 63-64 were rejected under 35 U.S.C. §103(a) as being unpatentable over *Abbott* in view of U.S. Publication No. 20040044989 (“*Vachuska*”).

Claims 5, 9-10, 24, 26, 35, 43 and 46 were rejected under 35 U.S.C. §103(a) as being unpatentable over *Abbott* in view of U.S. Publication No. 20030195923 (“*Bloch*”).

Claim 34 was rejected under 35 U.S.C. §103(a) as being unpatentable over *Abbott* in view of U.S. Patent No. 5,987,256 (“*Wu*”).

Claim 65 was rejected under 35 U.S.C. §103(a) as being unpatentable over *Abbott* in view of *Vachuska* and further in view of *Bloch*.

**Abbott in view of *Vachuska***

Claim 3 depends from claim 1. As discussed above, *Abbott* does not disclose several elements recited in claim 1. *Vachuska* does not provide the elements missing from *Abbott*. *Vachuska* discloses a source-code generator which uses a reflection mechanism to analyze source templates. *Vachuska* cannot be combined with *Abbott* to disclose the method recited in claim 1. In particular, *Vachuska* cannot modify *Abbott* to disclose building “optimized object code” by “identifying each application object in the serialized representation that has a unique identifier referring to an application object in the first object code file,” and “for each identified application object, copying the application object in the initialization code with the same unique identifier to the optimized object code file.” Therefore, claim 1 is not obvious over *Abbott* in view of *Vachuska*. Because claim 3 depends from claim 1, claim 3 is also not obvious over *Abbott* in view of *Vachuska*, and Applicants request that the Examiner withdraw this rejection.

Claim 40 depends from claim 38. For at least the same reasons discussed above regarding claim 3, claim 38 is not obvious over *Abbott* in view of *Vachuska*. Because claim 40 depends from claim 38, claim 40 is also not obvious over *Abbott* in view of *Vachuska*, and Applicants request that the Examiner withdraw this rejection.

Claims 62-64 depend from claim 62. For at least the same reasons discussed above regarding claim 3, claims 62-64 are not obvious over *Abbott* in view of *Vachuska*. Because claims 62-64 depend from claim 62, claims 62-64 are also not obvious over *Abbott* in view of *Vachuska*, and Applicants request that the Examiner withdraw this rejection.

**Abbott in view of Bloch**

Claims 5 and 9-10 depend from claim 1. As discussed above, *Abbott* does not disclose several elements recited in claim 1. *Bloch* does not provide the elements missing from *Abbott*. The Examiner cites *Bloch* for disclosing that a “runtime environment” may comprise a presentation renderer. Office Action, p. 31. Clarifying that the “runtime environment” recited in claim 1 is a virtual machine does not render claim 1 obvious over *Abbott* in view of *Bloch*. The combination of *Abbott* and *Bloch* still does not disclose building an “optimized object code” by “identifying each application object in the serialized representation that has a unique identifier referring to an application object in the first object code file,” and “for each identified application object, copying the application object in the initialization code with the same unique identifier to the optimized object code file.” Therefore, claim 1 is not obvious over *Abbott* in view of *Bloch*. Because claims 5 and 9-10 depend from claim 1, claims 5 and 9-10 are also not obvious over *Abbott* in view of *Bloch*, and Applicants request that the Examiner withdraw this rejection.

Claims 24 and 26 depend from claim 20. For at least the same reasons discussed above regarding claims 5 and 9-10, claim 20 is not obvious over *Abbott* in view of *Bloch*.

Because claims 24 and 26 depend from claim 20, claims 24 and 26 are also not obvious over *Abbott* in view of *Bloch*, and Applicants request that the Examiner withdraw this rejection.

Claim 35 depends from claim 33. For at least the same reasons discussed above regarding claims 5 and 9-10, claim 33 is not obvious over *Abbott* in view of *Bloch*. Because claims 35 depends from claim 33, claim 35 is also not obvious over *Abbott* in view of *Bloch*, and Applicants request that the Examiner withdraw this rejection.

Claims 43 and 46 depend from claim 38. For at least the same reasons discussed above regarding claims 5 and 9-10, claim 38 is not obvious over *Abbott* in view of *Bloch*. Because claims 43 and 46 depend from claim 20, claims 43 and 46 are also not obvious over *Abbott* in view of *Bloch*, and Applicants request that the Examiner withdraw this rejection.

#### *Abbott* in view of *Wu*

Claim 34 depends from claim 33. As discussed above, *Abbott* does not disclose several elements recited in claim 33. *Wu* does not provide the elements missing from *Abbott*. The Examiner cites *Wu* for clarifying that object code may include media assets. Office Action, p. 38. Clarifying that the object code disclosed in *Abbott* may include media assets does not disclose building an “optimized object code file” by “identifying each application object in the serialized representation that has a unique identifier referring to an application object in the first object code file,” and “for each identified application object, copying the application object in the initialization code with the same unique identifier to the optimized object code file.” Therefore, claim 33 is not obvious over *Abbott* in view of *Wu*. Because claim 34 depends from claim 33, claim 34 is also not obvious over *Abbott* in view of *Wu*, and Applicants request that the Examiner withdraw this rejection.

**Abbott in view of Vachuska and further in view of Bloch**

Claim 65 depends from claim 62. As discussed above, claim 62 is not obvious over *Abbott* in view of *Vachuska*. *Bloch* does not provide the elements missing from *Abbott*. The Examiner cites *Bloch* for disclosing that a “runtime environment” may comprise a presentation renderer. Office Action, p. 38. Clarifying that the “runtime environment” recited in claim 62 is a virtual machine does not render claim 62 obvious over *Abbott* in view of *Vachuska*, and further in view of *Bloch*. The combination of *Abbott*, *Vachuska* and *Bloch* still does not disclose building an “optimized object code” by “identifying each application object in the serialized representation that has a unique identifier referring to an application object in the first object code file,” and “for each identified application object, copying the application object in the initialization code with the same unique identifier to the optimized object code file.” Therefore, claim 62 is not obvious over *Abbott* in view of *Vachuska*, and further in view of *Bloch*. Because claim 65 depends from claim 62, claim 65 is also not obvious over *Abbott* in view of *Vachuska*, and further in view of *Bloch*, and Applicants request that the Examiner withdraw this rejection.

**Additional Remarks**

Based on the above amendments and these remarks, reconsideration of claims 1-15, 17, 19-31, 33-56 and 62-65 is respectfully requested.

The Examiner's prompt attention to this matter is greatly appreciated. Should further questions remain, the Examiner is invited to contact the undersigned attorney by telephone.

A Petition for Extension of Time has been submitted with this Response, extending the deadline to respond to the October 14, 2008 Office through February 14, 2009. Thus, this Response is timely filed.

The Commissioner is authorized to charge any underpayment or credit any overpayment to Deposit Account No. 501826 for any matter in connection with this response, including any fee for extension of time, which may be required.

Respectfully submitted,

Date: February 11, 2009

By: /Scott D. Sanford/  
Scott D. Sanford  
Reg. No. 51,170

VIERRA MAGEN MARCUS & DeNIRO LLP  
575 Market Street, Suite 2500  
San Francisco, California 94105  
Telephone: (415) 369-9660  
Facsimile: (415) 369-9665

[ssanford@vierramagen.com](mailto:ssanford@vierramagen.com)